Customizable Rating System and User List

Introduction

One of the core features of Anime List App project is to allow anime fans to customize their own ranking system in order to rate their animes/mangas in their user list. In contrast, other apps have only one or two predefined ranking systems for users to use which is fine in general but we're allowing the option to extend on it and make it something the super fans will enjoy. The vision is users are able to easily define what labels their ranking systems will employ, define scores and their descriptions, add weights, categories, and subscores, be able to share their ranking system for other users, and be able to interchange between ranking systems. At a high level, users are able to define their own criteria for scoring and thus be able to more succinctly and precisely rate a collection of animes or mangas, such as rating by genre, rating by movies, rating by release season, etc.

User Journey/Stories (Requirements)

1. As a user I can create a continuous, numerical rating system defining my own range for ratings.
2. As a user I can create a discrete rating system defining my own rating buckets.
3. As a user I can define subscores that add up to a particular overall rating in my rating system.
4. As a user I can switch between different rating systems.

Continuous Numerical Rating System

A continuous numerical rating system is simply a continuous range of scores. It's naturally ordered. A user can give any rating as long as it falls within the range. This allows some granularity too by fixing the decimal places. Subratings that add up to a final overall score are simple to implement: each rating gets n amount of subratings that are then averaged out (equally or weighted) to give a final score.

$$\sum_{i=1}^{n} x_i w_i = X$$

where x_i is the subscore within overall range, w_i is the weight of the subscore, and X is the final overall score. There must be at least two subratings. If “no subratings” are desired internally this means only one subrating: one with a weight of 100%.

Discrete Numerical Rating System

A discrete numerical rating system will follow similarly to a continuous one except there are specific rating buckets. For example, a discrete rating system with a range of 1-10 with 10 items will have discrete integers $1, 2, \ldots, 10$. Internally, a discrete rating system can be represented/calculated in a similar manner as continuous rating systems with the added step of “rounding up/down” the overall final rating.

So suppose a user wants 5 discrete rating buckets. Then it’s simply a continuous rating system with a range 1-5, along with discrete subratings, and calculated the same as continuous with the added step of “rounding up/down” the final score.

Discrete Labeled Rating System

A use case is where a user wants a non numerical rating buckets instead. An example being the ratings: S, A, B, C, D, F. Notice this is just a discrete set of integers, $1, 2, \ldots, 6$ that can be mapped one-to-one. So internally, this type of system will be implemented that same as a discrete numerical rating system with an extra final step of mapping the integers to a label using a function f.

Changing Rating Systems

Changing a rating system to another involves calculating the appropriate overall score. Suppose rating system A has max internal score max_A and a rating system B has max

internal score max_B. Given overall score in rating system A $score_A$ convert to the appropriate overall score in rating B.

$$score_B = \frac{score_A}{max_A} \times max_B$$

Note: Internal scores always start at 0 and is ≥ 0
An additional step after is to assign every subrating value to the overall internal score. So if rating system B has two subratings, just assign every subrating value to $score_B$.

High level steps rating system A to B

For every item:

1. Calculate $score_B$
2. Assign every subrating to $score_B$

User Lists

Currently, there is one user list per user where the AniList ID of the anime is stored along with a simple numerical rating. The userlist object is currently embedded into the user object.

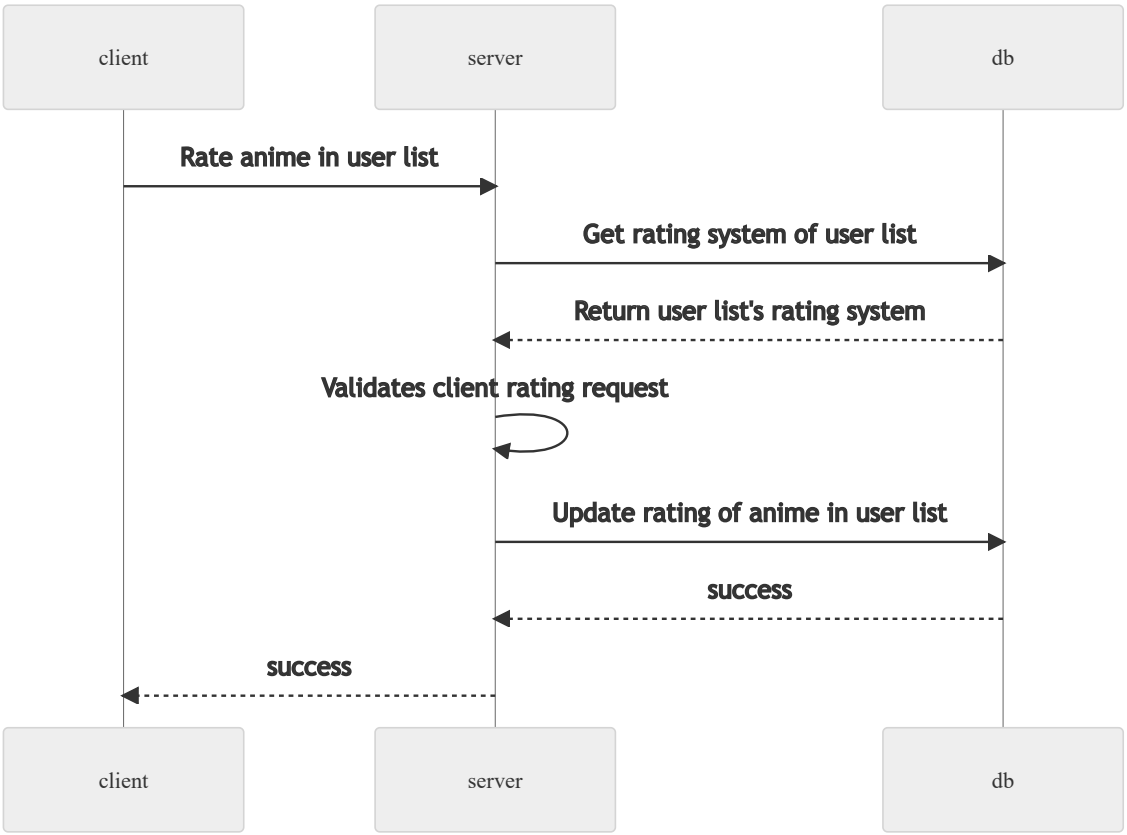
One of the planned features is to have multiple user lists each with their own selected ranking systems. For this, normalizing the user lists into a different collection in MongoDB is best.

Each user list will have a rating system embedded into the user list object if selected. A side effect to this duplication is when updating a rating system such as adding an extra subrating or an extra score or deleting, the changes are not propagated to the user list. Other alternatives is to have a reference, and whenever a rating system is updated or deleted then perform/handle the consequent actions to the ratings. For now the embedded approach is easier to implement with no side effects as the ratings should always be valid with the selected embedded rating system.

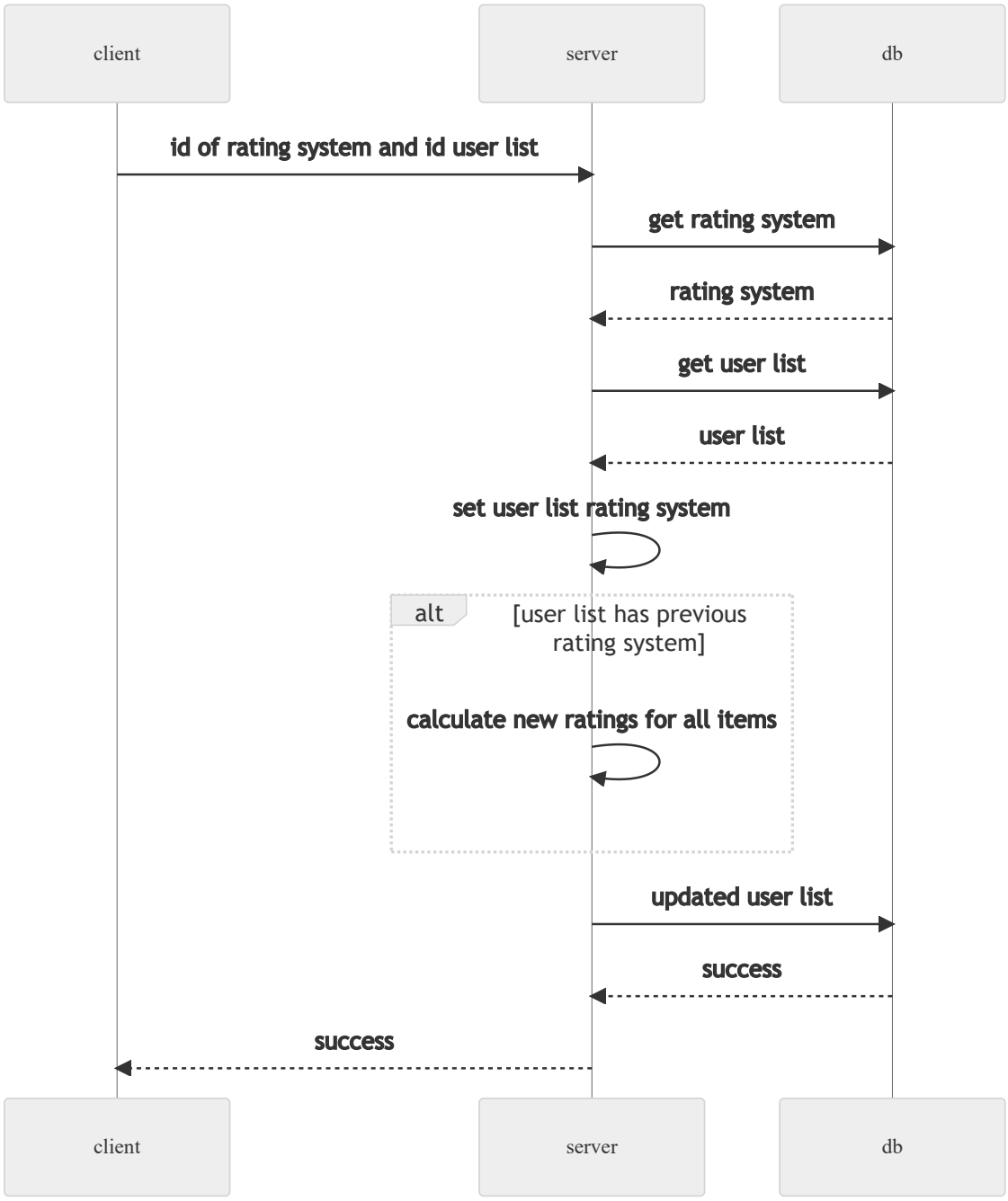
Dataflows

Some dataflows like creating/getting a userlist or rating systems are straightforward and not shown here. The following are more complex dataflows

Rating an anime in a userlist



Selecting a rating system for user list



Rating System Data Structure

The proposed solution to store rating systems is have Rating System documents in a separate collection.

Continuous Numerical Rating System

```
{
    "_id": ID // object id MongoDB
    "name": String // the name of this ranking set
    "ownerId": ID // object id the creator's user id
    "size": Int // the size of the range
    "type": "CONTINUOUS"
    "offset": Int // offset to add to the internal range
    "subRatings": [SubRating] // Array of subratings
}
```

SubRating datastructure

```
{
    "id": Int // identifier
    "name": String // name of the subrating
    "weight": Float // represents the weight of the subrating
}
```

A discrete rating system

```
{
    "_id": ID // object id MongoDB
    "name": String // the name of this ranking set
    "ownerId": ID // object id the creator's user id
    "size": Int // the size of the range
    "type": "DISCRETE"
    "subRatings": [SubRating] // Array of subratings
    "labels": [String] // Array of labels (1-to-1 function) where label[i] = t
}
```

Continuous Numerical Rating System Examples

A simple continuous rating system with where the display range is $1 - 10$.

```
{
    "_id": "7udu3838ri3ooe23"
    "name": "1-10 Rating"
    "ownerId": "89eke92384ideoi"
    "size": 10
    "type": "CONTINUOUS"
    "offset": 1
    "subRatings": [
        {
            "id": 0 // identifier
            "name": "score"
            "weight": 1.0
        }
    ]
}
```

Note: there is only one subrating with weight 1.0 since there are no actual subratings.

A continuous rating system with display range $1 - 10$ with subratings: Beginning, Middle, End weighted equally

```
{
    "_id": "7udu3838ri3ooe23"
    "name": "Beginning, Middle, End 1-10 Rating"
    "ownerId": "89eke92384ideoi"
    "size": 10
    "type": "CONTINUOUS"
    "offset": 1
    "subRatings": [
        {
            "id": 0 // identifier
            "name": "Beginning"
            "weight": 0.3333333
        }
        {
            "id": 1 // identifier
            "name": "Middle"
            "weight": 0.3333333
        }
        {
```

```
            "id": 2 // identifier
            "name": "End"
            "weight": 0.3333333
        }
    ]
}
```

Discrete Rating System examples

A simple discrete rating system of range $1, 2, ..., 10$

```
{
    "_id": "7udu3838ri3ooe23"
    "name": "1-10 Discrete Rating"
    "ownerId": "89eke92384ideoi"
    "size": 10
    "type": "DISCRETE"
    "subRatings": [
        {
            "id": 0 // identifier
            "name": "score"
            "weight": 1.0
        }
    ]
    "labels": ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10"]
}
```

A simple discrete rating system of following ordered (low to high) labels: 👎, 😐,👍 with two subratings: “Visual” and “Sound”

```
{
    "_id": "7udu3838ri3ooe23"
    "name": "1-10 Discrete Rating"
    "ownerId": "89eke92384ideoi"
    "size": 3
    "type": "DISCRETE"
    "subRatings": [
        {
            "id": 0
            "name": "Visual"
```

```
            "weight": 0.5
        }
        {
            "id": 1
            "name": "Sound"
            "weight": 0.5
        }
    ]
    "labels": ["👎", "😐", "👍"]
}
```

User List data structure

User List data structure

```
{
    "_id": ID // object id
    "Name": String // the name of user list
    "ownerId": ID // the user id of who owns this list
    "ratingSystem": RatingSystem // the embedded selected rating system
    "items": [UserListItem] // the items in this list
}
```

UserListItem data structure

```
{
    "mediaID": Int // AniList mediaID
    "rating": UserListRating // the rating
}
```

UserListRating data structure

```
{
    "displayRating": String // overall display name rating
    "rating": Int // overall internal rating
    "subRatings": [
```

```
            {
                "id": Int // identifier
                "name": String // display name
                "rating": Int // internal rating for this subrating
            },
            ...
        ]
    }
```

User List data structure example

```
{
    "_id": "873ukjd73uie3" // object id
    "Name": "My Main List" // the name of user list
    "ownerId": "jdi387riuhk34" // the user id of who owns this list
    "ratingSystem": {
        "_id": "7udu3838ri3ooe23"
        "name": "1-10 Rating"
        "ownerId": "89eke92384ideoi"
        "size": 10
        "type": "CONTINUOUS"
        "offset": 1
        "subRatings": [
            {
                "id": 0
                "name": "score"
                "weight": 1.0
            }
        ]
    }
    "items": [
        {
            "mediaID": 12738 // AniList mediaID
            "rating": {
                "displayRating": "10" // overall display name rating
                "rating": 9
                "subRatings": [
                    {
                        "id": 0
                        "displayRating": "10"
                        "rating": 9
```

```
                        }
                    ]
                }
            }
        ]
    }
```

API

CreateUserList(input: CreateUserListInput): Mutation

GetUserList(input: GetUserListInput): Query

AddUserListRatingSystem(input: AddUserListRatingSystemInput): Mutation

AddUserListItem(input: AddUserListItemInput): Mutation

RateUserListItem(input: RateUserListItemInput): Mutation

CreateRatingSystem(input: CreateRatingSystemInput): Mutation

GetRatingSystem(input: GetRatingSystemInput): Query